**Docker Guide for Beginners**

Authored by **Sineesha Sukumaran**

This guide is crafted to help newcomers gain a clear, practical understanding of Docker — from installation and containerization basics to networking, volumes, and secure deployments. Whether you're a developer, student, or IT professional, this document provides the foundation to build and run containerized applications with confidence.

Essential Concepts and Hands-on Introduction to Docker

Table of Contents

[1. What Is Docker? 2](#_Toc204520277)

[Why Use Docker? 2](#_Toc204520278)

[Docker Architecture 2](#_Toc204520279)

[Docker Client & REST API 2](#_Toc204520280)

[What is Docker Engine? 3](#_Toc204520281)

[What is Docker Image? 3](#_Toc204520282)

[What is a Dockerfile? 3](#_Toc204520283)

[What is a Container? 3](#_Toc204520284)

[What is Docker Compose? 3](#_Toc204520285)

[What is a Docker Volume? 3](#_Toc204520286)

[What is Docker Hub? 3](#_Toc204520287)

[What is the Docker Daemon? 4](#_Toc204520288)

[What is Port Mapping in Docker? 4](#_Toc204520289)

[What is Docker CLI? 4](#_Toc204520290)

[How is Docker different from normal software installations? 5](#_Toc204520291)

[Examples of Use Cases 5](#_Toc204520292)

[2. Getting started with Docker 5](#_Toc204520293)

[For Windows Users 5](#_Toc204520294)

[For macOS Users 5](#_Toc204520295)

[For Linux (Ubuntu Example) 5](#_Toc204520296)

[Verify Docker Installation (All Systems) 6](#_Toc204520297)

[Troubleshooting Tips 6](#_Toc204520298)

[3. First Steps with Docker CLI 6](#_Toc204520299)

[4. Creating Your Own Image with a Dockerfile (Windows) 6](#_Toc204520300)

[How to create a DockerFile 7](#_Toc204520301)

[Explanation with an Example 8](#_Toc204520302)

[Troubleshoot 10](#_Toc204520303)

[Best practices: 11](#_Toc204520304)

[Docker Quick Reference: All Commands & Code Snippets 11](#_Toc204520305)

[**Conclusion**  13](#_Toc204520306)

# 1. What Is Docker?

Docker is a tool that package applications and all their dependencies into a container – **a lightweight, portable unit.** It is like a box that holds your app and everything needs to run. No matter where you open the box, the app works the same way, **without any setup problems.**

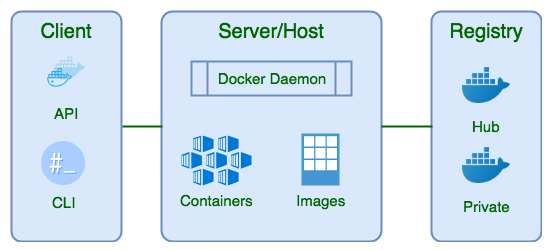
It ensures **consistency, saves system resources**, and allows you to **run apps the same way on any computer**—whether it's your laptop or a server in the cloud.

Unlike virtual machines (heavyweight and separate OS stacks), containers share the main system's core but keep each app **separate and safe from others**.

## Why Use Docker?

* **Works the same everywhere** – No more problems like “It works on my computer but not on yours.”
* **Fast and light** – Uses less memory and starts quickly, unlike bulky virtual machines.
* **Easy to move around** – You can run it on your laptop, a friend's computer, or in the cloud without any extra setup.

## Docker Architecture



## Docker Client & REST API

The **Docker Client** is like a remote control—you use to give commands to Docker (like docker run or docker build).

Behind the scenes, it talks to **Docker Daemon** (the one that runs your containers) using the **Docker REST API**, which is just a way to send instructions over the internet.

***Example***: When you type ***docker ps*** to see running containers, the client asks the daemon for the list using the REST API.

## What is Docker Engine?

Docker Engine is the tool that runs containers. You install it on your computer, and it **helps build and manage these containers.**

## What is Docker Image?

Think of a Docker image like a *recipe* or a *blueprint*. It tells Docker **exactly how to build a container.**For example an image might say:

“Use Python 3.10, install this app, and copy these files.”

When Docker runs an image, it creates a container from it—just like baking a cake from a recipe.

## What is a Dockerfile?

A Dockerfile is a simple text file that tells Docker **how to build your own image.**  
It lists step-by-step instructions like:

Start from Python → Copy files → Install packages → Run the app

You can start, stop, or delete it anytime.

## What is a Container?

A container is like a lightweight package that includes your app, code, libraries, and settings. It doesn’t include a full operating system—just **what your app needs to run**.

## What is Docker Compose?

A Docker Compose is a tool that **lets you run multiple containers together** as one app.  
You write a small config file, and it starts everything with one command.  
Perfect for full-stack apps (like backend + frontend + database).

## What is a Docker Volume?

A Docker volume **lets you save data outside the container**. Even if the container **stops or is deleted**, the data stays safe. Useful for databases, logs, or user files.

When you create a volume, you’re telling Docker:  
***“Keep this folder's data safe in a special place, not inside the temporary container.”***

Then you **connect (mount)** that volume to a folder **inside** the container.

## What is Docker Hub?

Docker Hub is like the *App Store* or *Play Store*—but for Docker.  
It’s an online place where you can:

* **Download ready-to-use images** (like Ubuntu, Python, PostgreSQL, etc.)
* **Share your own images** publicly or keep them private for your team.

***Example***: Want to run MySQL? Just pull its image from Docker Hub with one command.

## What is the Docker Daemon?

The Docker Daemon (also called dockerd) is the **background service** that does all the heavy lifting in Docker.

Think of it like the **engine** **running under the hood**—it manages:

* Building images
* Running containers
* Monitoring and cleaning up
* Handling requests from the Docker CLI or Docker Desktop

When you type a Docker command like **docker run**, you’re giving instructions to the Docker Daemon. The daemon then makes it happen behind the scenes.

So, even though you don’t see it, the Docker Daemon is **always working silently** in the background to run your containers smoothly.

## What is Port Mapping in Docker?

Port mapping **connects a port inside the container to your real machine.  
*Example***:

Imagine you have a house (your computer) with a locked room (a container) inside it.

Now, your friend (a user or browser) wants to visit that room — but they can’t go directly inside unless you open a door.

Port mapping is like saying:

***docker run -p 8080:5000 my-app***

***Means***: “Hey, use door number 8080 on my house, and I’ll guide you straight to room number 5000 inside.”

In real terms: You can open localhost:8080 to use the app running inside on port 5000.

## What is Docker CLI?

The Docker CLI (Command Line Interface) is the tool you use to talk to Docker using text commands in **your terminal or command prompt.**

## How is Docker different from normal software installations?

Normally, when you install software, it depends on your system settings and installed packages. But with Docker, everything your app needs is bundled in a container, so **it works the same on any machine.**

## Examples of Use Cases

* Developers work in teams, so everyone uses the same setup.
* Running websites, databases, or AI models in isolated environments.
* Testing apps quickly without setting everything up manually.

# 2. Getting started with Docker

Docker works on Windows, macOS, and Linux. Here’s how to install and test it on your system:

*Go to*: <https://www.docker.com/products/docker-desktop>

**Download Docker Desktop.**

## For Windows Users

* Run the installer and follow the steps (Next → Next → Finish).
* After installation, restart your computer if a message appears asking you to do so.
* Open Docker Desktop (it may take a few seconds to start).

You should see the Docker whale icon in the system tray (bottom-right corner).

## For macOS Users

* Open the downloaded .dmg file and drag Docker into Applications.
* Open Docker Desktop from Launchpad.
* Allow permission if prompted (e.g., system extensions).

Wait for Docker to start with. You’ll see the whale icon on the bar menu.

## For Linux (Ubuntu Example)

For Ubuntu, you **don’t need Docker Desktop**. You can install **Docker Engine + CLI** directly using commands.

Open your terminal and run:

* ***sudo apt update***
* ***sudo apt install docker.io***
* ***sudo systemctl start docker***
* ***sudo systemctl enable docker***

To Avoid using sudo every time (Optional):

* ***sudo usermod -aG docker $USER***

*This will add your user to the Docker group, so you don’t have to type sudo every time you run Docker commands.*

Then log out and log back in, or reboot.

## Verify Docker Installation (All Systems)

Make sure Docker Desktop is running, then open your terminal or command prompt and type:

***docker run hello-world***

If Docker is working correctly, you'll see:  
***"Hello from Docker! This message shows that your installation appears to be working correctly."***

## Troubleshooting Tips

* Make sure Docker is running (check for the Docker whale icon).
* On Linux, try using sudo if needed:

***sudo docker run hello-world***

* On Windows, enable WSL 2 if prompted (Docker will guide you).
* Restart your computer after installation if things don’t work.

# 3. First Steps with Docker CLI

Key commands to start using Docker on the command line:

* ***docker run image*** # creates and starts a container.
* ***docker ps*** # lists running containers.
* ***docker ps -a*** # lists all containers (including stopped ones).
* ***docker images*** # shows downloaded images.
* ***docker stop <id>*** # Stop a running container
* ***docker rm <id>*** # Remove a container

The <id> refers to the **Container ID** or **name.**

# 4. Creating Your Own Image with a Dockerfile (Windows)

A Dockerfile doesn’t use a normal programming language like Python or Java. Instead, it uses a **special set of instructions** created by Docker. These instructions tell Docker **how to build an image** step by step — like a **recipe** for setting up an app.

*Some common instructions include:*

* ***FROM*** – Choose the starting point (like the base image)
* ***WORKDIR*** – Sets the working folder inside the container (like choosing the default location to work in).
* ***COPY*** – Copy files into the image
* ***RUN*** – Run setup commands (like installing software)
* ***EXPOSE*** – Specify which port the container will use to communicate with the outside world (like saying: “Hey, I’ll be listening on port 80!”)
* ***CMD*** – Tell Docker what to do when the container starts

Think of it like giving your computer a **to-do list** in simple steps and Docker follows it to prepare your app environment.

## How to create a DockerFile

1. Open any **text editor** (like Notepad, VS Code, or even Notepad++).
2. Paste Docker instructions (example below)

A screenshot of a computer program

AI-generated content may be incorrect.

1. Save the file as: Dockerfile (**no file extension**).
2. Place it in the folder with your app files.

## Explanation with an Example

**Command 1:**

***FROM mcr.microsoft.com/windows/servercore:ltsc2022***

This tells Docker to start from **a Windows Server base image** instead of Linux (like Ubuntu). It provides a Windows environment inside the container.

**Command 2:**

***WORKDIR /app***

This sets **/app** as the folder where all your app files will go and where commands will run from. Think of it like saying: “CD into this folder first.”

**Command 3:**

***COPY . /app***

This copies all your project files (from the folder where your Dockerfile is) into the /app folder inside the container.

**Command 4:**

***RUN powershell.exe -Command "write-output 'Hello from Windows Docker container!' "***

Thiscommand tells your computer to open PowerShell (a command-line tool) and run simple instructions to print the message

**Command 5:**

***EXPOSE 8080***

This opens port 8080 inside the container so your app can be accessed from your computer (like visiting <http://localhost:8080>).

We use port **8080** because:

* Port **80** is usually already used by your system or browser.
* Port **8080** is like a **spare gate** for web apps — easy to remember and not commonly blocked.
* Many developers pick it by default for testing or local apps.
* Instead of 8080, beginners can also use ports like **3000, 5000, or 8000** — they are commonly used for web app development and are usually free to use on your system.

**Command 6:**

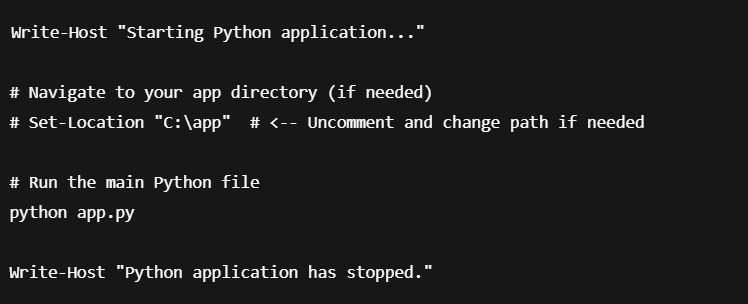
***CMD ["powershell.exe", "./start.ps1"]***

it tells Docker:  
***“Every time this container runs, automatically run that script for me.”***

You can replace start.ps1 with any other script or command to control what your app or service does when the container launches.

start.ps1 is just a text file that you create but with a .ps1 extension so Windows knows it's a PowerShell script. A .ps1 file is written in the **PowerShell scripting language**. It is like a **robot assistant** that can press buttons, move files, open apps, and follow smart instructions automatically, without you clicking anything!

***Example start.ps1*:**

****

Place this file **in the same folder** as your Dockerfile if you're using it in your image build.

After creating Dockerfile and Start.ps1, run this command in powersheell :

* ***docker build -t <image-name> <path-to-dockerfile>***
* ***docker run <image-name>***
* ***docker run -p 8080:8080 <image-name>***

**Explanation**

* ***-t <image-name>*** Tag your image with a name (replace with your own, you can choose any name you like.).
* ***<path-to-dockerfile>*** Usually **“.”** for current folder, or a path to the Dockerfile.
* ***docker run <image-name>*** Runs the built image.
* ***docker run -p 8080:8080 <image-name>***Runs the built image and connects your computer’s port 8080 to the container’s port 8080, so you can access the app at http://localhost:8080.

*Also, make sure* ***Docker Desktop is turned on*** *(running) before you start.*

**Summary**

This Dockerfile:

* Using a Windows base image
* Copies your app files into the container
* Run your startup script when the container launches

So, you don’t have to open the container and start things manually — it just works, like pressing a play button.

**Notes:**

* **If you're using Windows containers**, tell Docker to run a PowerShell script like **start.ps1** and use **Windows-style paths** with backslashes like this: *folder\script.ps1*
* **If you're using Linux containers, use a** script like **start.sh** and **Linux-style paths** with slashes like this: *folder/script.sh.*
* You can use **Python files (.py) in both cases**. Just run them with python or python3, based on what your container supports.

## Troubleshoot

**1.**Check your version, run this in PowerShell:

***systeminfo | findstr /B /C:"OS Name" /C:"OS Version"***

If it says **Windows Home**, Windows containers won't work. Using a Windows base image like:

***FROM mcr.microsoft.com/windows/servercore:ltsc2022***

will throw errors on Windows Home.Windows containers require **Windows** **Pro, Enterprise, or Education** editions.

**Solution**: Use a Linux base image instead, like:

***FROM python:3.10-slim***

***WORKDIR /app***

***COPY . /app***

***RUN echo "Hello from Linux Docker container!"***

***CMD ["python", "start.py"]***

Use ***CMD ["python", "start.py"]*** instead of start.*ps1 because Linux containers don’t support PowerShell (.ps1) scripts they run Python or Bash commands instead.*

**Then run:**

***docker build -t <image-name> <path-to-dockerfile>***

***docker run <image-name>***

**2.** By default, Docker looks for a file named **Dockerfile** (case-sensitive, no extension).  
 If you use a different name, you must specify it with the -f flag:

***docker build -f <CustomDockerfileName> -t <image-name> .***

This works because Docker on Windows Home uses Linux containers via WSL 2.

## Best practices:

* Keep your app files in a separate folder.
* Use a .dockerignore file to skip unnecessary files (like logs or .git).
* Write commands in the right order to **reuse cache** and make builds faster.

## Docker Quick Reference: All Commands & Code Snippets

**Basic CLI Commands**

docker run hello-world  
docker ps # list running containers  
docker ps -a # list all containers  
docker images # list available images  
docker stop <container-id>  
docker rm <container-id>

**Build & Run Docker Images**

docker build -t my-image .  
docker run my-image  
docker run -p 8080:8080 my-image

**Sample Dockerfile (Linux Base)**

FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN echo "Hello from Linux Docker container!"

EXPOSE 8080

CMD ["python", "start.py"]

**Sample Dockerfile (Windows Base)**

FROM mcr.microsoft.com/windows/servercore:ltsc2022

WORKDIR /app

COPY . /app

RUN powershell.exe -Command "write-output 'Hello from Windows Docker container!'"

EXPOSE 8080

CMD ["powershell.exe", "./start.ps1"]

**Linux Installation Commands**

sudo apt update  
sudo apt install docker.io  
sudo systemctl start docker  
sudo systemctl enable docker  
sudo usermod -aG docker $USER

**Docker Compose (Basic Example)**

version: "3"  
services:  
 web:  
 image: my-web-app  
 ports:  
 - "8080:80"  
 db:  
 image: postgres:latest  
 volumes:  
 - db-data:/var/lib/postgresql/data  
volumes:  
 db-data:

## Conclusion

We have now learned the basics of Docker - what it is, how to install it, use key commands, and build your first container using a Dockerfile. This is just the beginning.

👉 In the next part, we’ll cover Docker Compose, volumes, Understanding Docker Networking, and best practices to take your skills further.

*Until then, happy learning and all the best on your Docker journey!*

## 📚 Additional Resources

You can access all the guides (including future parts) from my GitHub repo:  
🔗 [**https://github.com/sineesha/your-data-handbook.git**](https://github.com/sineesha/your-data-handbook.git)

Star it ⭐ to stay updated!